Containers, Container Orchestration, Architecture

Benjamin Zhang <bzh>
Based on Brian Sang’s Fall 2019 slides

Where we left off

- At scale, failure is common
- Solved problem of configuring large number of machines

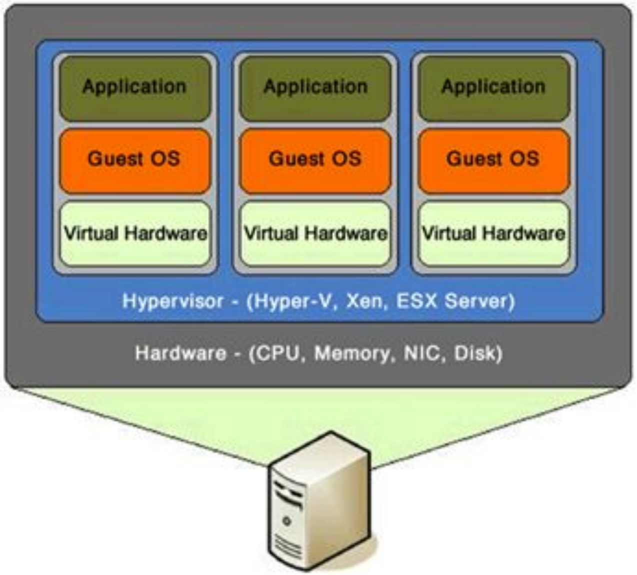
This week

- How to make our applications run on large systems?
  - Seamless handover if a server keels over
  - How does Google/FB/etc. have nearly perfect uptime?

Containers

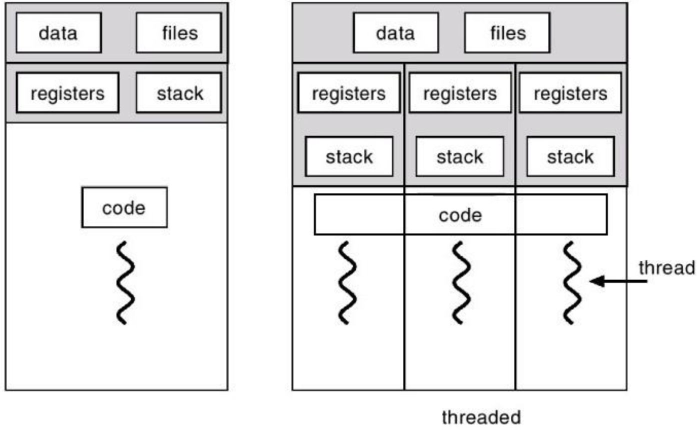
What are VMs exactly?

- Virtual Machines
- Emulates a physical machine using software
- Typical setup: hypervisor runs multiple VMs
- Provides isolation
- Some bloat: Need different Guest OS and emulation of Virtual Hardware for each applica
- Takes some time to boot up

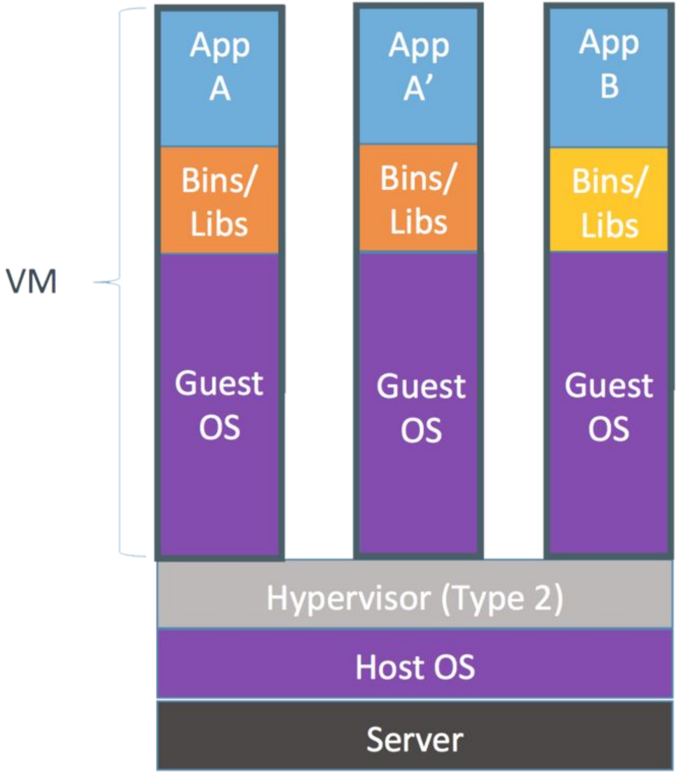


What is a process, exactly?

- Take CS162
  - Best class
  - Don’t argue with me
- An executing program
- Independent virtual address space, program, CPU state, program state, file descriptor #s...
- ...and more. Everything you need to run a program (shell, web browser, pubg, etc.)
  - Includes other binaries and libraries which may be dynamically linked

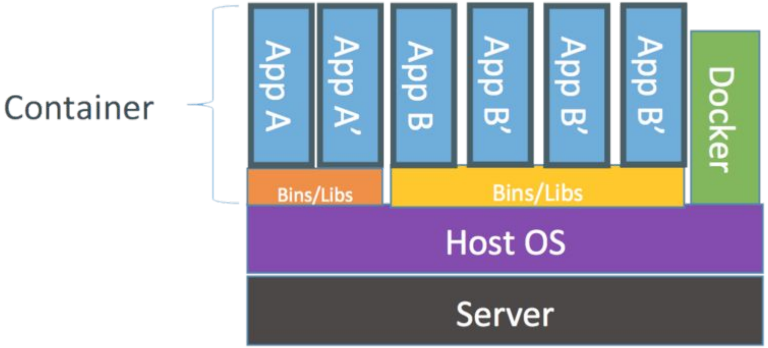


Containers - between VMs and processes?

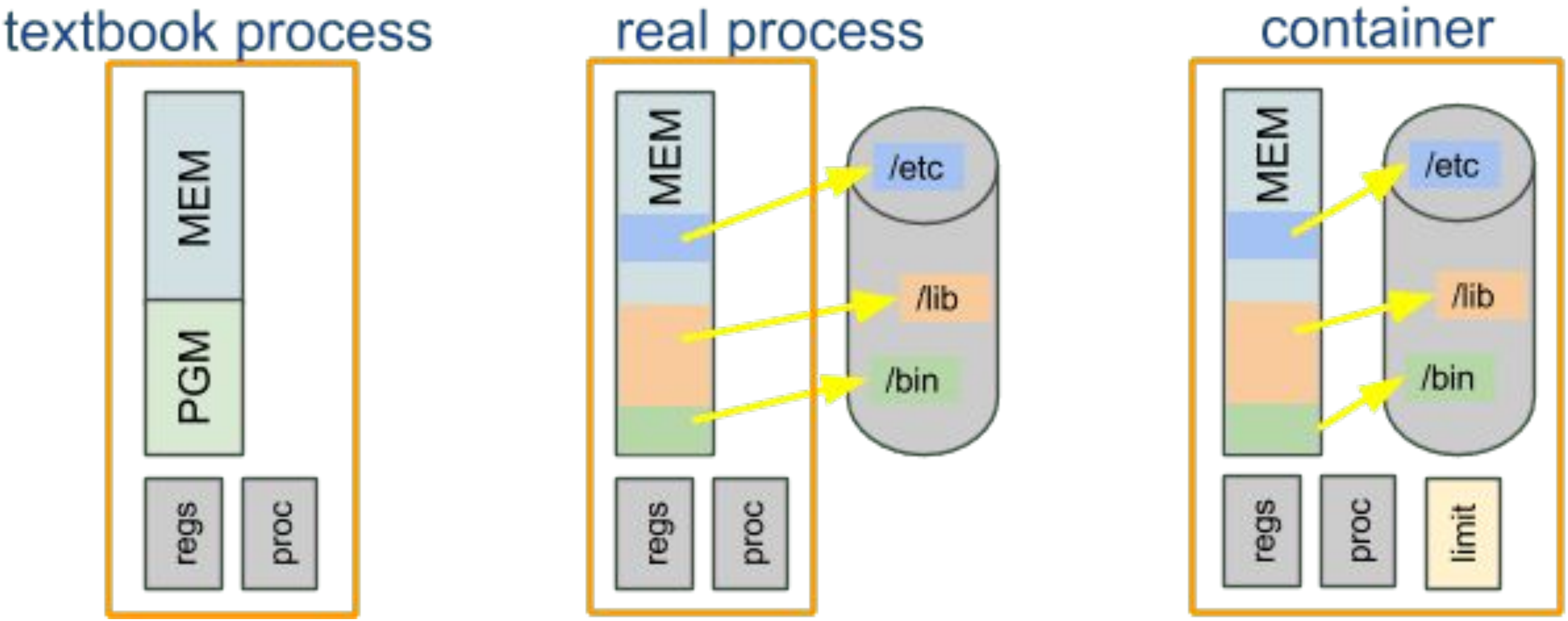


Containers are isolated, but share OS and, where appropriate, bins/libraries

...result is significantly faster deployment, much less overhead, easier migration, faster restart



Containers vs. Processes



textbook process

real process

container

Containers — Big Picture!

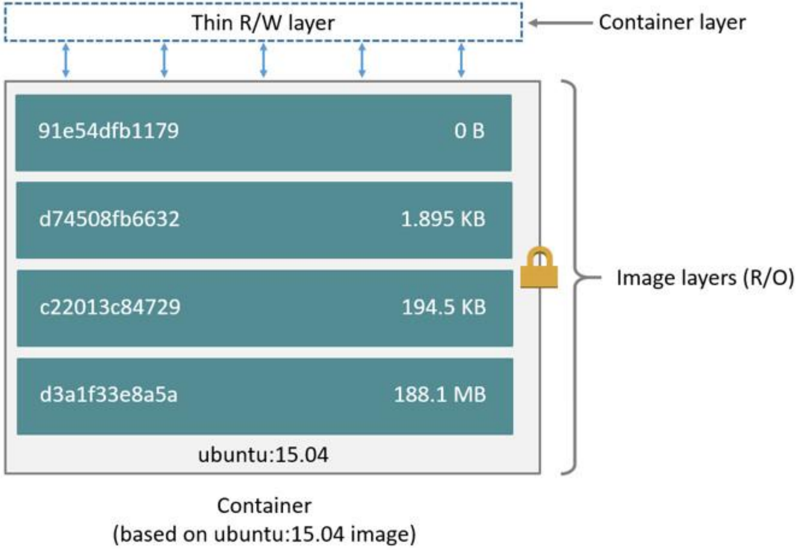
- Often compared with VMs, but are more like processes with bundled environments
- Provide similar isolation
  - Much less than VMs, however! For this reason, we often run containers inside VMs still (but can run >1 container per VM)
- Much faster to boot up than VMs
- Easy to package applications into containers
- Distribute built containers (*images*) to quickly deploy
- Common container runtimes: Docker, rkt, LXC

Docker Concepts - Images and Containers

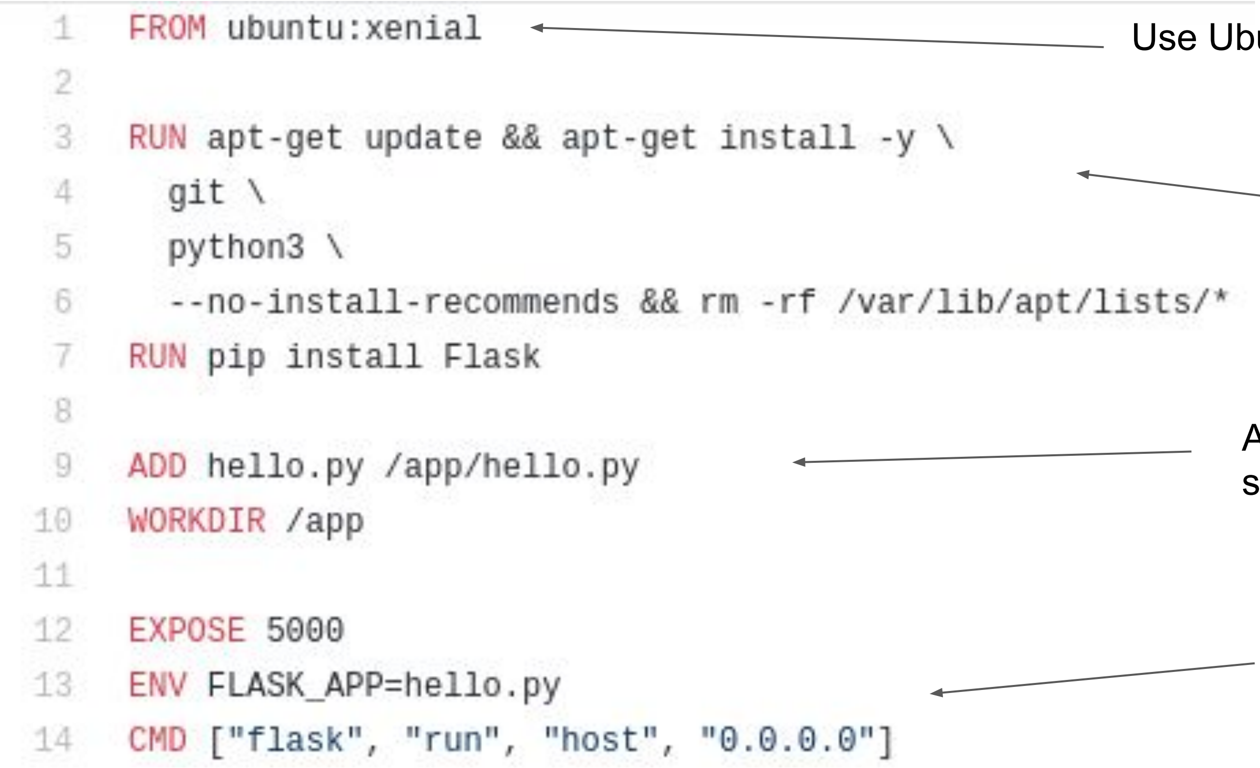
- Image: an executable package that includes everything needed to run an application
  - the code, a runtime, libraries, environment variables, and configuration files.
- Container: A runnable/running instance of an image
- Similar to Class vs. Object concept in OOP
- Can pull/push built images from a registry, e.g. DockerHub
  - Or a private registry

Docker Concepts - Building an Image, Dockerfiles

- Images need to be built
- Typically use a Dockerfile to specify how to build an image
- Images are built in layers
  - Like an onion :shrek:
  - Allows for images based off the same layer to be built faster
- Keep each layer minimal



Dockerfile Example



```
FROM ubuntu:xenial

RUN apt-get update && apt-get install -y \
  git \
  python3 \
  --no-install-recommends && rm -rf /var/lib/apt/lists/*
RUN pip install Flask

ADD hello.py /app/hello.py
WORKDIR /app

EXPOSE 5000
ENV FLASK_APP=hello.py
CMD ["flask", "run", "host", "0.0.0.0"]
```

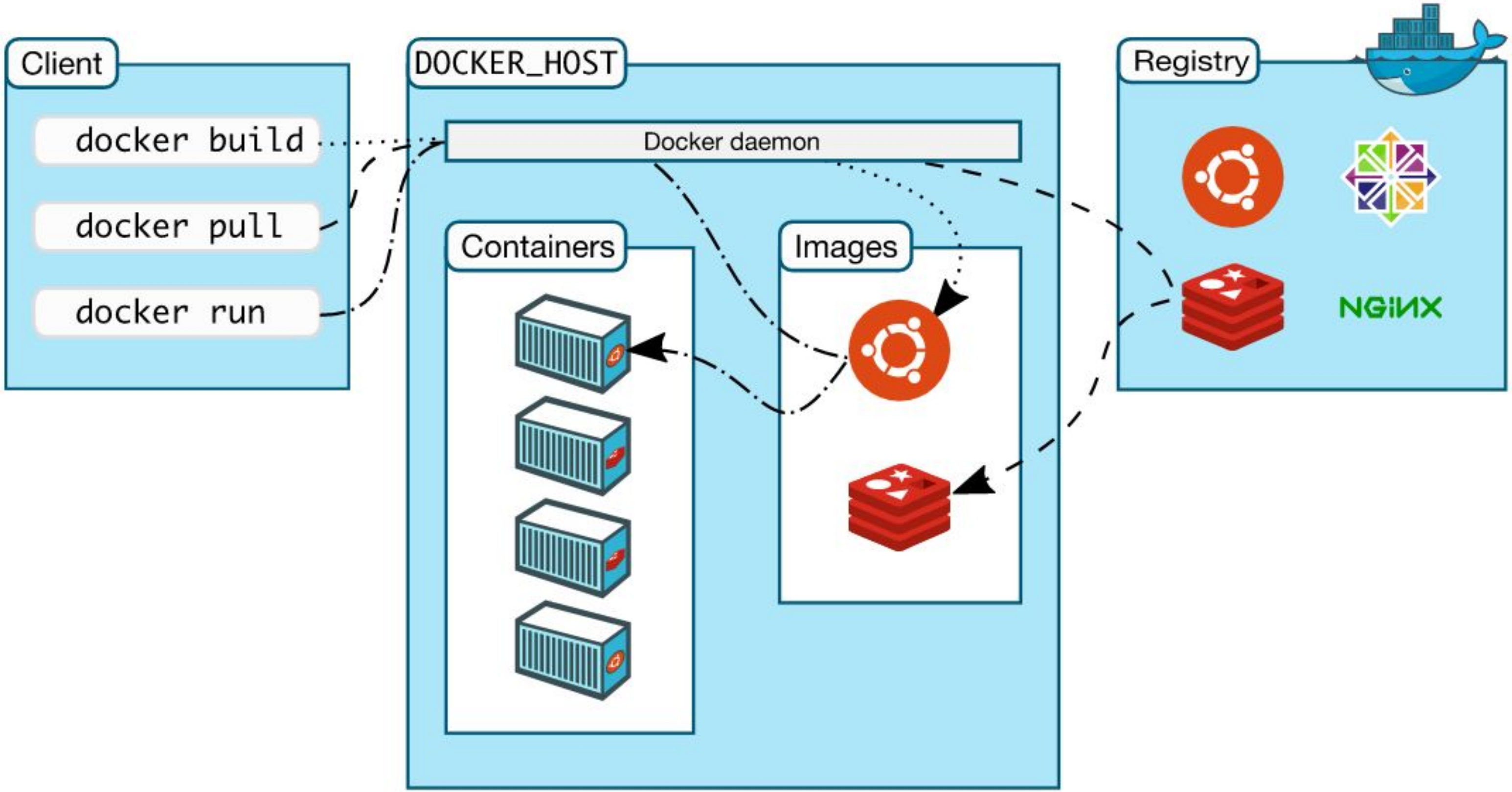
Use Ubuntu as base image

Update the system and install requisite packages
(delete extra files to keep things minimal!)

Add local files to run the server to our container

Run the webserver!

The Docker daemon



Container Orchestration

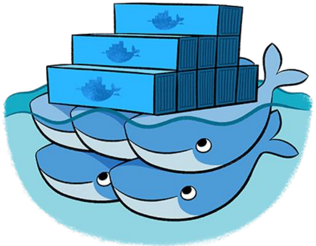
Container Orchestrators

- So we have the capability of putting all of our apps into nice containers, how do we actually use them?
- Container Orchestrators help us make sure we have the right amount of containers we want, that they are on the proper machines, restart them when they die to handle failure, etc.
- Use **Distributed Systems Magic™** to recover from failures; multiple masters, distributed key/value stores, and more
- Examples: Kubernetes, Mesos+Marathon, Docker Swarm

kubernetes

MESOS

MARATHON

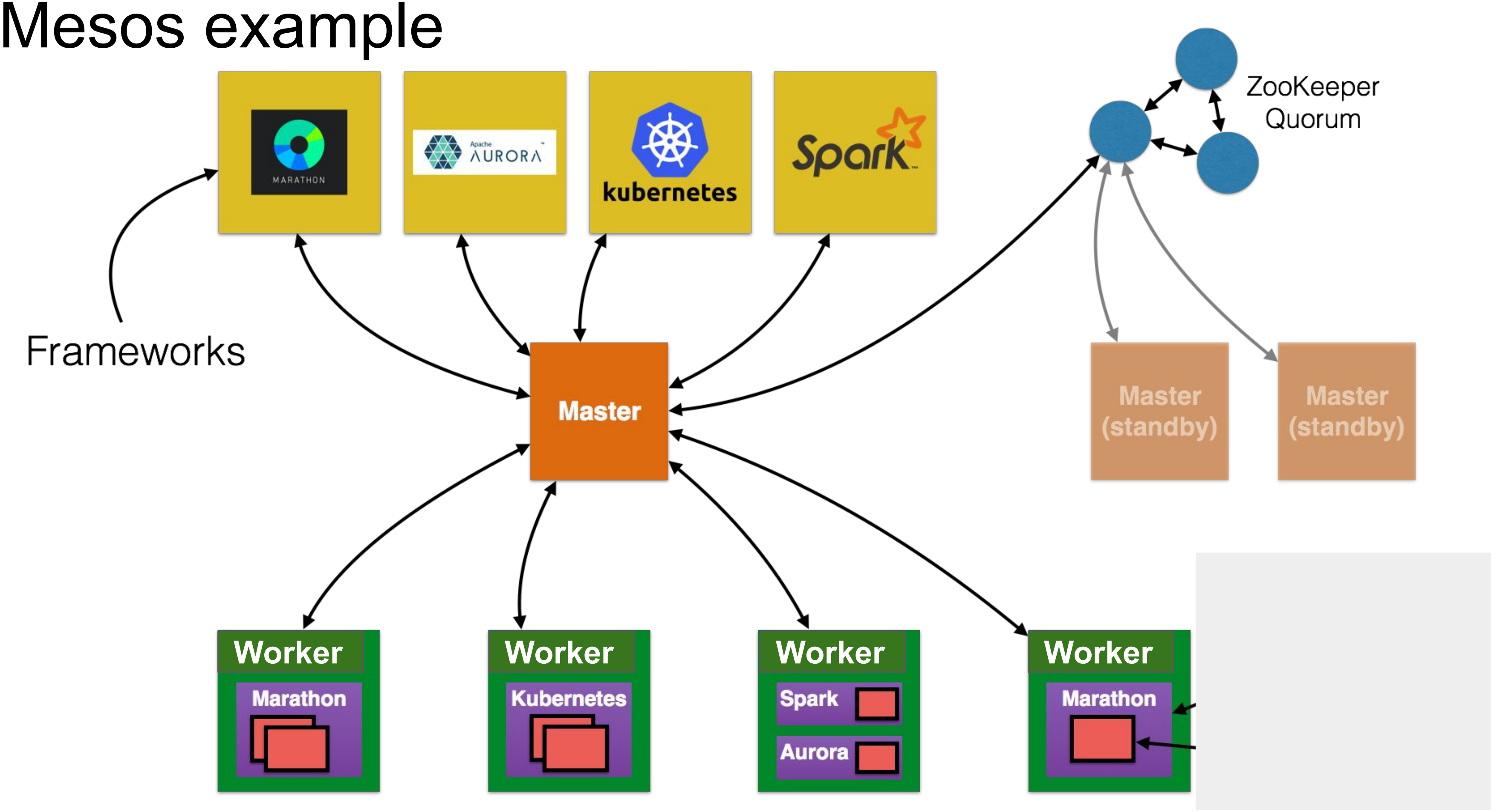


A taste of Distributed Systems Magic™

Typical Structure - Master/Worker Architecture

- Multiple masters in case one fails
  - Maintain quorum using etcd, Zookeeper (distributed key/value stores)
    - Use Paxos, Raft to maintain consensus (badass two phase commit)
- Some sort of scheduler
  - Place containers on machines
- Some sort of cluster manager
  - Scale up/down machines and have them join the cluster

Mesos example



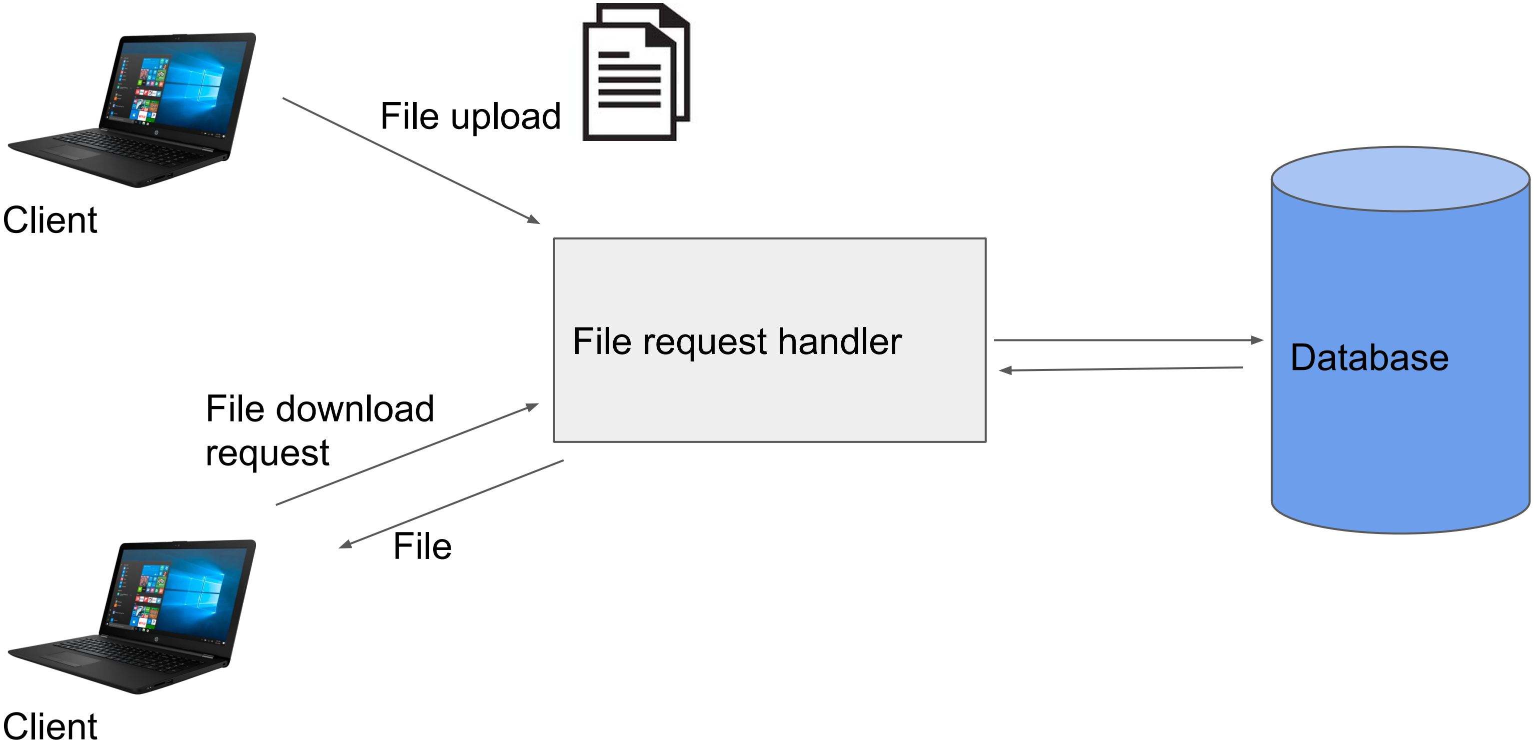
Distributed Systems is hard

- Need to make sure all nodes are actually doing what a master/coordinator says
    - Problem: a node could fail and stop responding at any time!
    - Could fail right after it says “Yes, I’m about to do this”
- All of these algorithms are just for getting nodes to AGREE on something
    - 2PC - too slow
    - Paxos - notoriously hard to understand
    - Raft - supposedly easier to understand. I don’t really understand it (haven’t read paper yet lol)
- Don’t roll your own crypto? Don’t write your own consensus algorithm
    - Just use etcd (raft) or zookeeper (custom algorithm, zab)

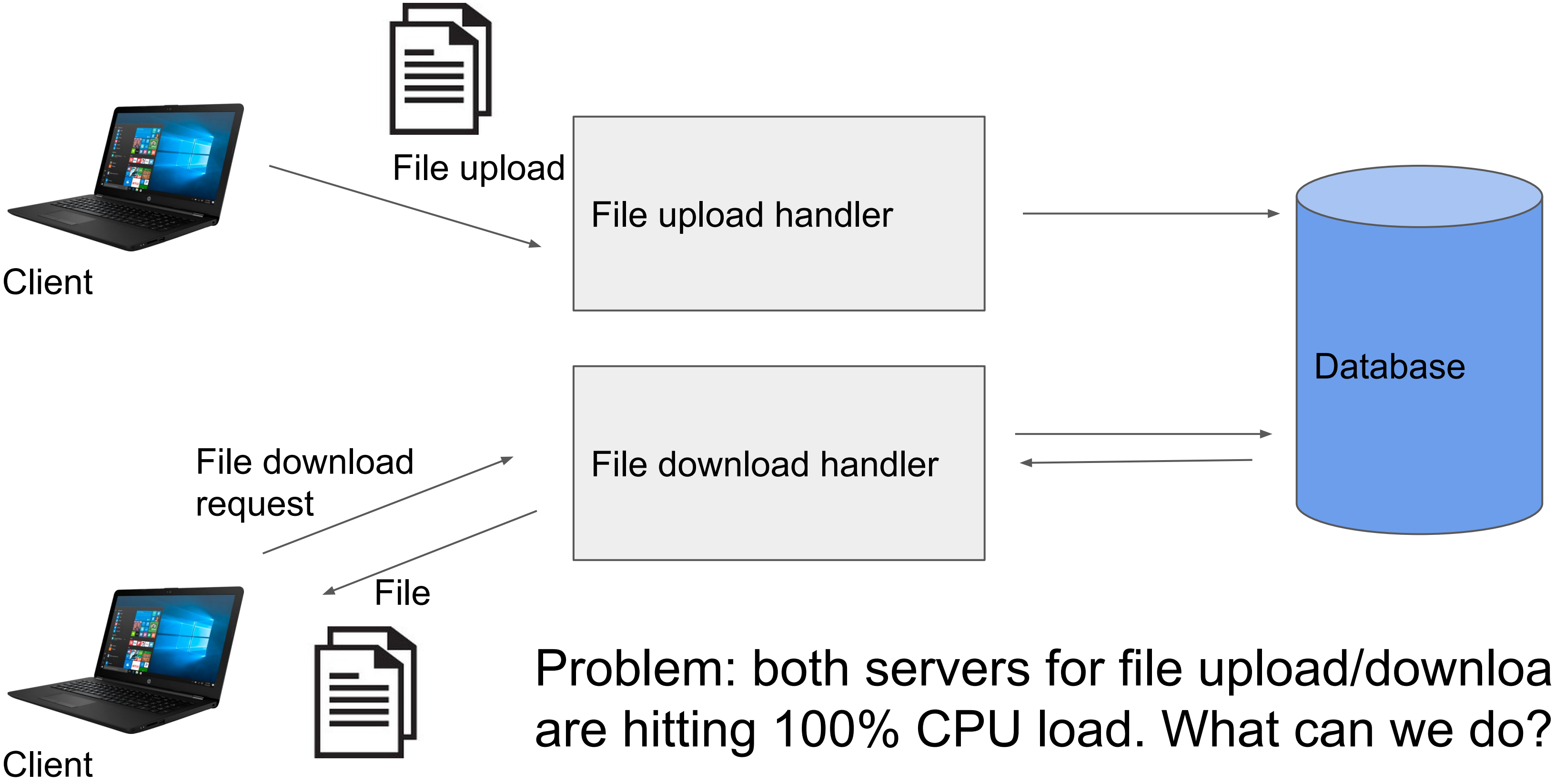
Architecture and Design

Don’t read ahead! This part is meant to be interactive

Basic File storage service

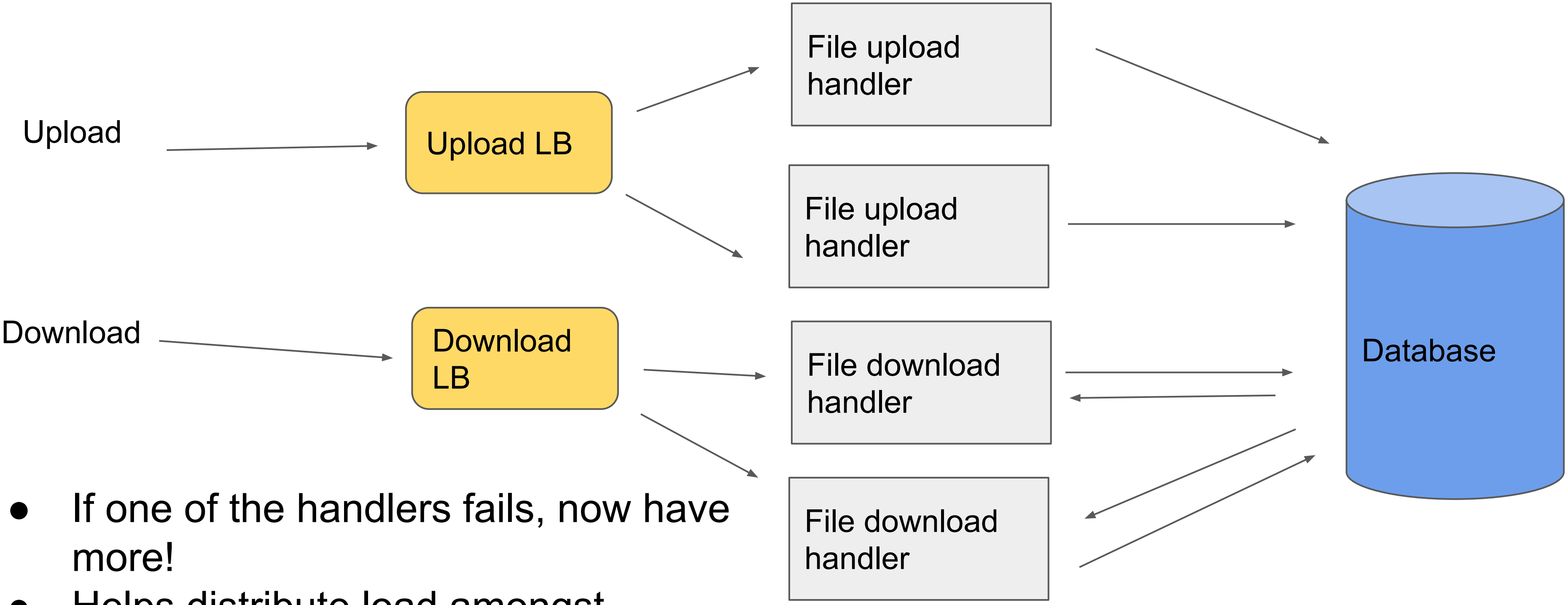


Divide into separate services



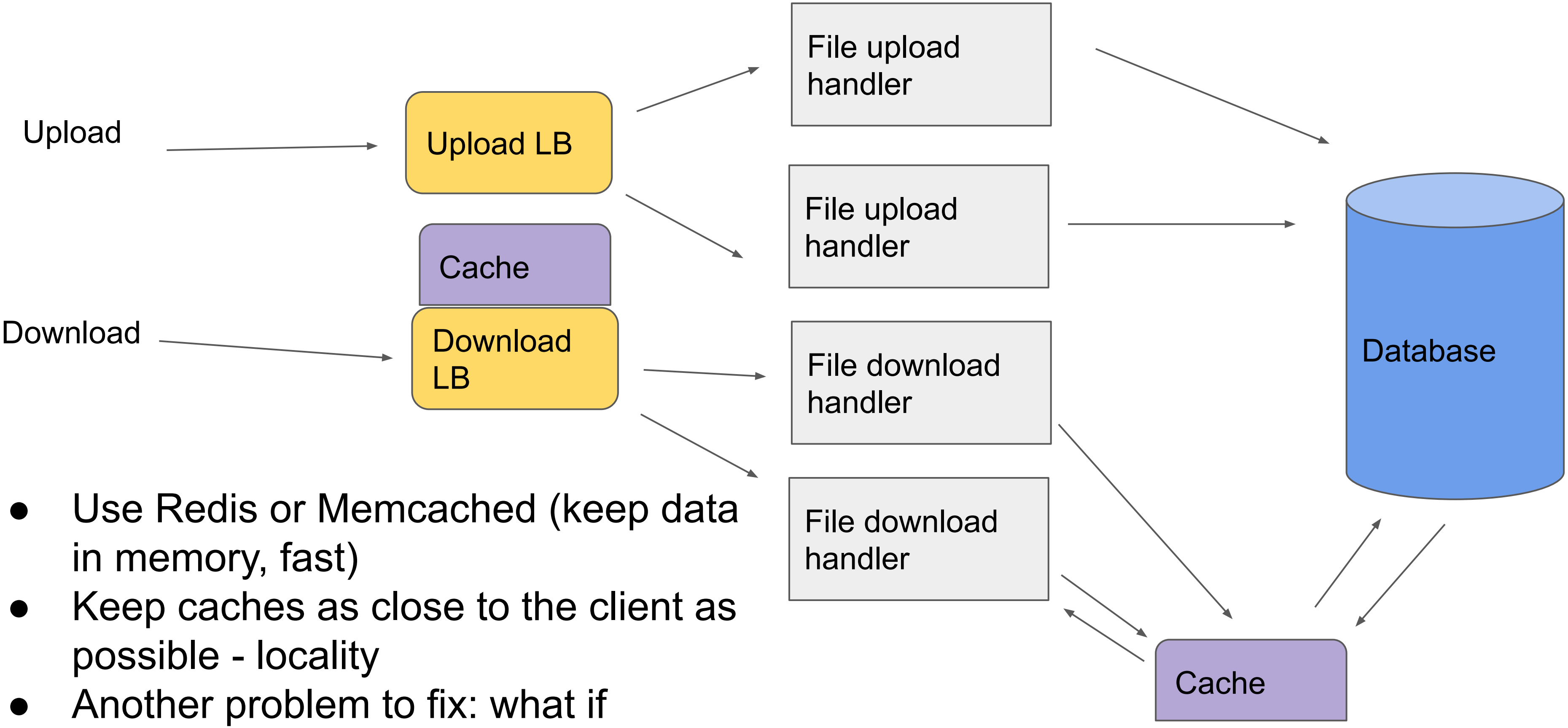
Problem: both servers for file upload/download are hitting 100% CPU load. What can we do?

Improvement 1: More handlers



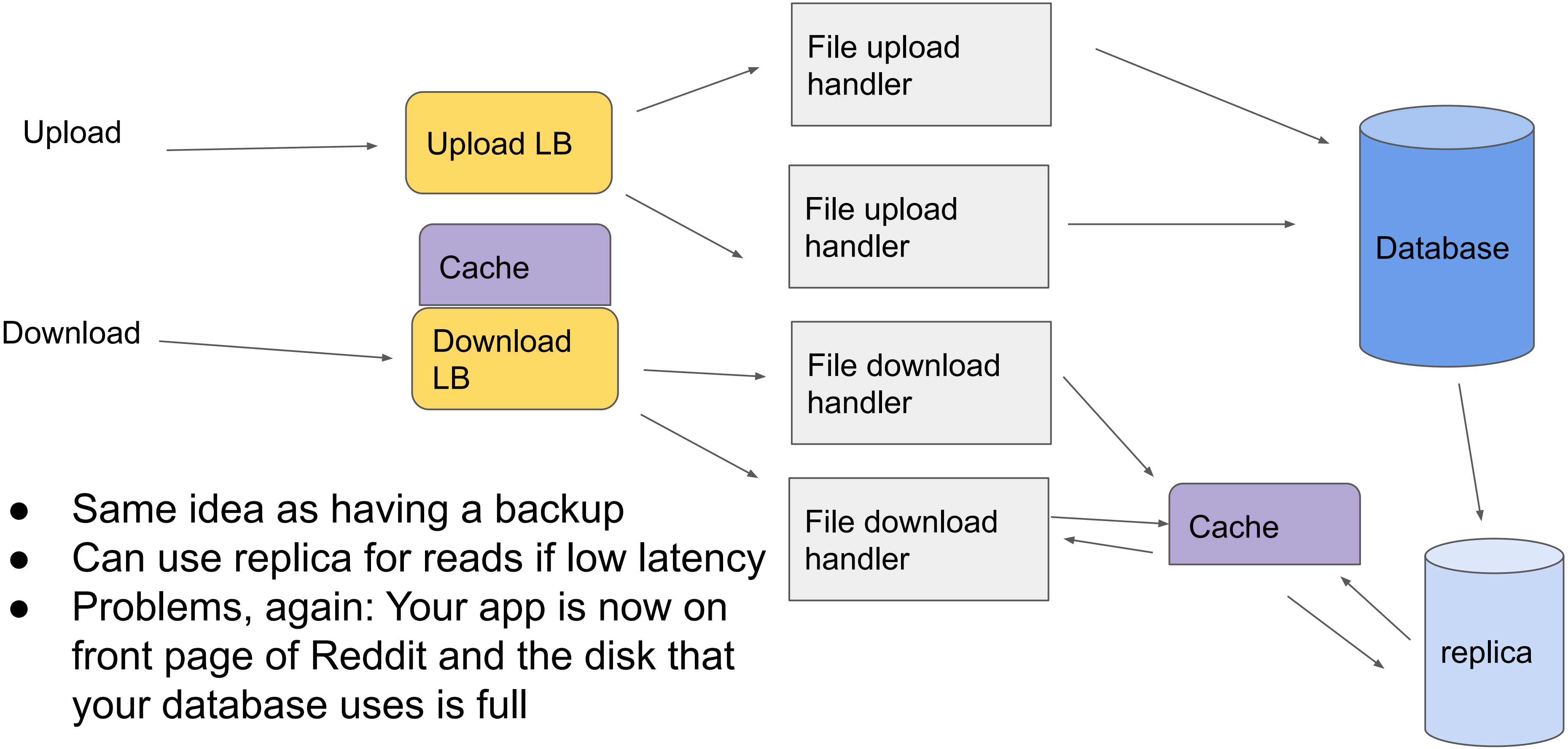
- If one of the handlers fails, now have more!
- Helps distribute load amongst handlers - scale out
- New problem: Database disk I/O is maxing out, what do we do now?

Improvement 2: Caching!



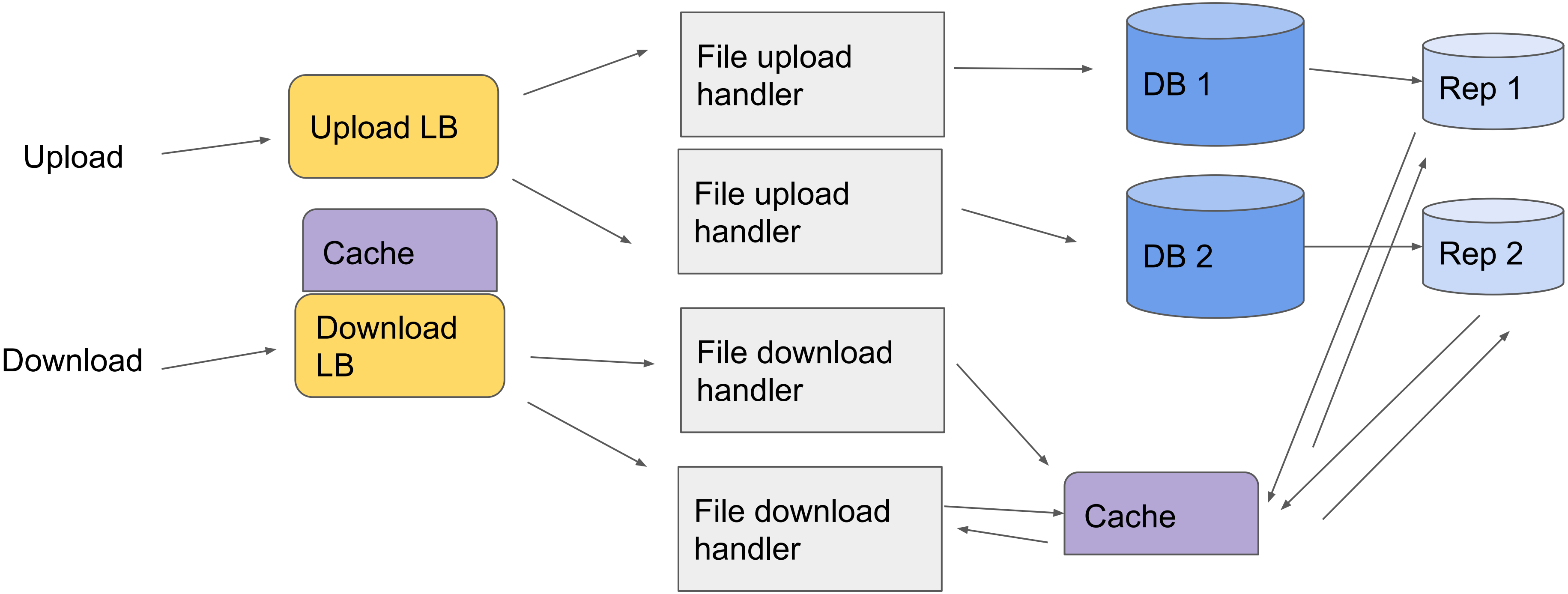
- Use Redis or Memcached (keep data in memory, fast)
- Keep caches as close to the client as possible - locality
- Another problem to fix: what if database goes down?

Improvement 3: Replication



- Same idea as having a backup
- Can use replica for reads if low latency
- Problems, again: Your app is now on front page of Reddit and the disk that your database uses is full

Improvement 4: Partition



- Split database into multiple machines
- Need to route requests to correct DB
- More complexity so that’s kinda bad
- Now we can scale out application servers layer and data layer

Recap

- With scale, new problems
- Tradeoffs tradeoffs tradeoffs
- Scale horizontally when possible
  - But, this introduces complexity
- UNIX philosophy: small sharp tools
  - Small, sharp services scale better
  - Introduces complexity when you need to combine them